Maximum Flows and Paremetric Shortest Paths in Planar Graphs [3]

Filip Koprivec

Faculty of Mathematics and Physics, IMFM

June 2020

Maximum flow problem

Given a

- Directed graph (V, E) (simple)
- Edge capacities (weights): $c : E \to \mathbb{R}^+$ (maximum available edge flow)

and $s, t \in V$

Find a feasible flow $f : E \to \mathbb{R}^+$

- $f(e) \leq c(e)$ Satisfies capacity constraints
- Conserves flow:

$$\forall v \notin \{s, t\}.\ \sum_{u \to v} f(u \to v) = \sum_{v \to w} f(v \to w)$$

Find one with

$$max(\sum_{s \to v} f(s \to v))$$

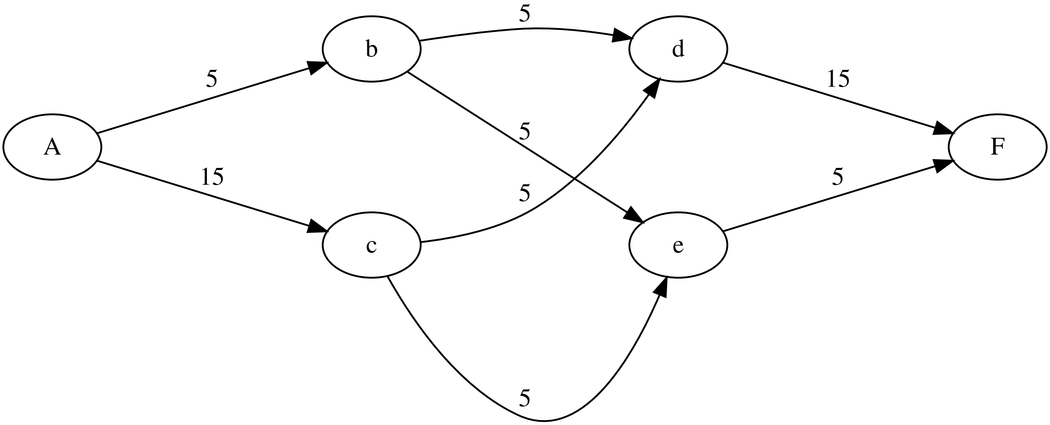


Figure: Input graph

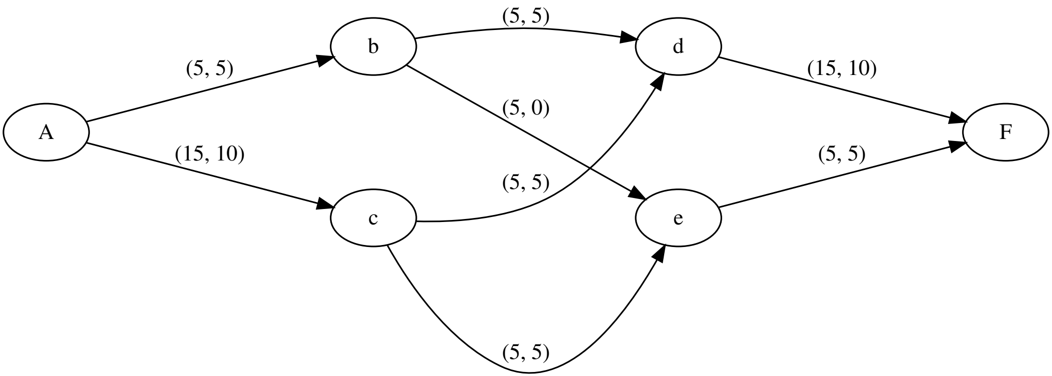


Figure: Possible solution: 15

Applications

Direct practical applications

- Max flow (routing problems)
- Huge impact in data centers
- Various aspects of resource management and scheduling ("baseball elimination" problem, project resource allocation)

First studied by Soviets before WW2, used by USA military planers during the cold war to asses military capabilities [6].
The solution for East European railway (10 stations, ~70 connections) was produced by hand.

(General) Solutions

- Classical Ford-Fulkerson algorithm [4] ($O(Ef)$) and Edomnds-Karp ($O(VE)$), Dinic ($O(VE \log E)$ using dynamic trees).
- Max-flow min-cut theorem (Max-flow and min-cut are a reformulation as a primal-dual linear programs)

Most direct applications of maximum flow problems can be modeled as planar graphs, so the question is, whether better solutions exists in such cases.

Planar graphs

- Computing flow with a fixed value is equivalent to computing a SSSP in a particular dual Graph G^* [5].
- In 2009 An $O(n \log n)$ algorithm was described for arbitrary directed planar graphs [1] (using clockwise/counterclockwise cycling to saturate network).
- Problem is intricately connected to the shortest paths in dual graph -> most of the solutions rely on fast SSSP algorithms
- SSSP can be computed in linear time (with no negative weights), when s and t are on the same face, the problem is linear.

A simpler derivation of $O(n \log n)$ algorithm [1]

- Reformulate max-flow as a parametric SSSP in dual graph
- Solve SSSP in the dual
- Devise an upper bound of number of changes in the dual
- Construct a higher genus graph, that achieves upper bound

Dual graph

WLOG: Assume that all edges have a reverse pair

$$(u \to v)^* := u^* \uparrow v^* \qquad u^* \uparrow v^* := (u \to v)^*$$

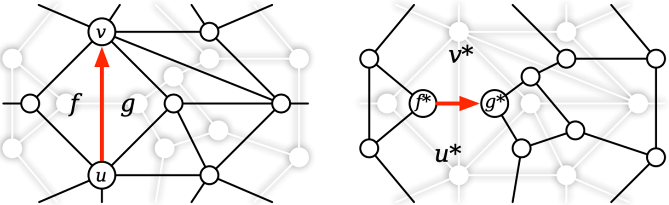


Figure: Dual graph [3]

Flow

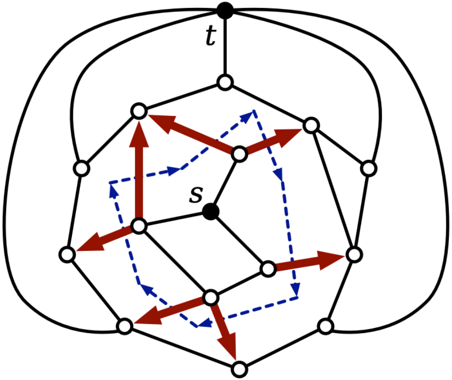
$$\phi : E \to \mathbb{R} \qquad\qquad \phi(e) = 0, e \notin E$$

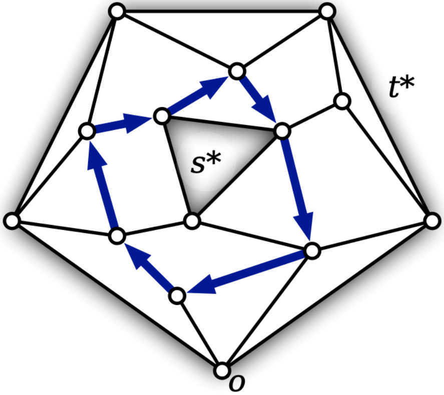
$$\phi(e) = -\phi(rev(e)) \qquad\qquad \sum_{w} \phi(v \to w) = 0$$

Feasible if $\phi(e) \leq c(e)$. Flow saturates an edge if $c(e) = \phi(e)$. For a cut $C \subseteq E$: $\phi(C) = \sum_{e \in C} \phi(e)$.
By the max-flow min cut, if C is saturated, then $\phi(C)$ is max flow.

Reformulation

Cut is called a *Cocycle* if the dual produces a simple cycle





Crossing number

For a arbitrary (fixed) path P in G.

$$\pi(e) := \begin{cases} 1 & e \in P \\ -1 & e \notin P \\ 0 & \text{otherwise} \end{cases}$$

For a cycle C in dual, this counts the number of left to right crosses minus the number of right to left crosses.
$\pi(C) \in \{-1, 0, 1\}$ for any cocycle C and $\pi(C) = 1 \iff C$ is a (s, t)-cut.

Crossing number

For a arbitrary (fixed) path P in G.

$$\pi(e) := \begin{cases} 1 & e \in P \\ -1 & e \notin P \\ 0 & \text{otherwise} \end{cases}$$

For a cycle C in dual, this counts the number of left to right crosses minus the number of right to left crosses.
$\pi(C) \in \{-1, 0, 1\}$ for any cocycle C and $\pi(C) = 1 \iff C$ is a (s, t)-cut. Apply Jordan curve, and do case analysis on side location of t, s. Cycle is polygon.

Residual flow

For $\lambda \in \mathbb{R}$ consider $\lambda\pi$.
Take the residual graph with $c(\lambda, e) = c(e) - \lambda\pi(e)$.
Flow is feasible $\iff$ $c(\lambda, e) \leq 0$.
Consider the residual dual network $c(\lambda, e^*) := c(\lambda, e)$.
There exists a feasible (s, t) flow $\iff$ the residual network G^* doesn't contain a negative cycle.

Residual flow

For $\lambda \in \mathbb{R}$ consider $\lambda\pi$.
Take the residual graph with $c(\lambda, e) = c(e) - \lambda\pi(e)$.
Flow is feasible $\iff$ $c(\lambda, e) \leq 0$.
Consider the residual dual network $c(\lambda, e^*) := c(\lambda, e)$.
There exists a feasible (s, t) flow $\iff$ the residual network G^* doesn't contain a negative cycle. If there is a cycle:

$$c(\lambda, C^*) = \sum_{e \in C} c(\lambda, e) = \sum_{e \in C} c(e) - \lambda\pi(C) < 0$$

$\pi(C) = 1$ and C is a (s, t)-cut with a capacity smaller than λ.

No negative cycles

Root a tree of SSSP at arbitrary $o \in G^*_\lambda$

$$\phi(\lambda, e) = d(\lambda, \mathit{head}(e^*)) - d(\lambda, \mathit{tail}(e^*)) + \lambda\pi(e)$$

For an arbitrary v, the edges leaving v define directed (simple) cycle in G^* and the d cancels out -> the net value of sink in 0. This is a valid flow.
$\mathit{slack}(\lambda, e^*) := d(\lambda, \mathit{tail}(e*)) - d(\lambda, \mathit{head}(e^*)) + c(\lambda, e) = c(e) - \phi(\lambda, e)$, which is feasible by [4].

No negative cycles

Root a tree of SSSP at arbitrary $o \in G^*_\lambda$

$$\phi(\lambda, e) = d(\lambda, head(e^*)) - d(\lambda, tail(e^*)) + \lambda\pi(e)$$

For an arbitrary v, the edges leaving v define directed (simple) cycle in G^* and the d cancels out -> the net value of sink in 0. This is a valid flow.
$slack(\lambda, e^*) := d(\lambda, tail(e*)) - d(\lambda, head(e^*)) + c(\lambda, e) = c(e) - \phi(\lambda, e)$, which is feasible by [4].
Since path is arbitrary, for a fixed λ one can calculate the flow in $O(n)$.

Solution

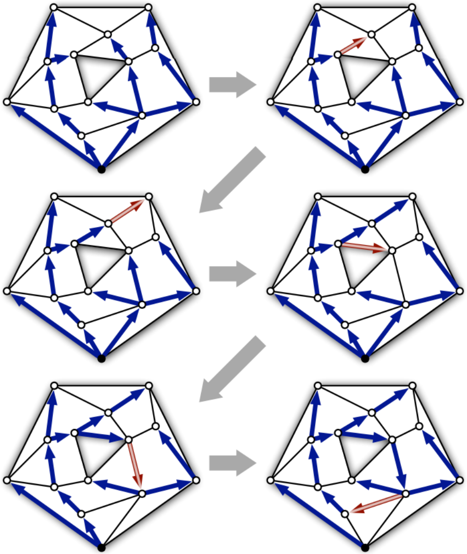
Find the largest value λ that produces no negative cycles.
Find a λ, that induces a zero cycle. This is exactly the parametric shortest path problem.
WLOG: assume, that all the SSSP tree is always unique (standard perturbations, lexicographical...)

For $\lambda \in [0, \lambda_{max}]$ a SSSP tree is well defined (and unique) except for some critical values.

For $\lambda \in [0, \lambda_{max}]$ a SSSP tree is well defined (and unique) except for some critical values.
Edges of the tree point outwards, and $slack(\lambda, p)$ (in dual) has exactly one incoming vertex.

For $\lambda \in [0, \lambda_{max}]$ a SSSP tree is well defined (and unique) except for some critical values.
Edges of the tree point outwards, and $slack(\lambda, p)$ (in dual) has exactly one incoming vertex.
Of special interest are *tense* edges $slack(\lambda, e^*) = 0$, they always lie on the tree except on critical values of λ (pivots), where they get replaced by some other.
Loose edges in original graph are the ones where dual edges are not tense (spare capacity).

For $\lambda \in [0, \lambda_{max}]$ a SSSP tree is well defined (and unique) except for some critical values.
Edges of the tree point outwards, and *slack*(λ, p) (in dual) has exactly one incoming vertex.
Of special interest are *tense* edges *slack*$(\lambda, e^*) = 0$, they always lie on the tree except on critical values of λ (pivots), where they get replaced by some other.
Loose edges in original graph are the ones where dual edges are not tense (spare capacity).



Complexity

Initialize the spanning tree of loose edges L_λ, SSSP predecesors for dual vertices and slack values for dual edges. This can be done in $O(n \log n)$
Crucial part is use of self adjusting top tree to efficiently update spanning tree in $O(\log n)$ time.

Complexity

Initialize the spanning tree of loose edges L_λ, SSSP predecesors for dual vertices and slack values for dual edges. This can be done in $O(n \log n)$

Crucial part is use of self adjusting top tree to efficiently update spanning tree in $O(\log n)$ time.

Final complexity: $O(n \log n) + O(N \log n)$ where N is the number of pivot points.

Complexity

Initialize the spanning tree of loose edges L_λ, SSSP predecesors for dual vertices and slack values for dual edges. This can be done in $O(n \log n)$
Crucial part is use of self adjusting top tree to efficiently update spanning tree in $O(\log n)$ time.
Final complexity: $O(n \log n) + O(N \log n)$ where N is the number of pivot points. If t is incident to the outer face, each dual edge pivots in the shortest path at most once. (Borradaile proved O(n) pivots in general case).

Complexity

Initialize the spanning tree of loose edges L_λ, SSSP predecesors for dual vertices and slack values for dual edges. This can be done in $O(n \log n)$

Crucial part is use of self adjusting top tree to efficiently update spanning tree in $O(\log n)$ time.

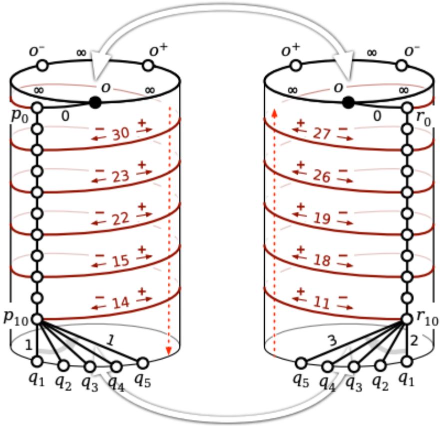
Final complexity: $O(n \log n) + O(N \log n)$ where N is the number of pivot points. If t is incident to the outer face, each dual edge pivots in the shortest path at most once. (Borradaile proved O(n) pivots in general case).

Consider an infinite strip graph with two pivot paths with indices $i < j$ Concatenate $o_{-i} - - > p_0$ and $o_j - - > p_0$ and loop. q_0 is both in and out of the loop.

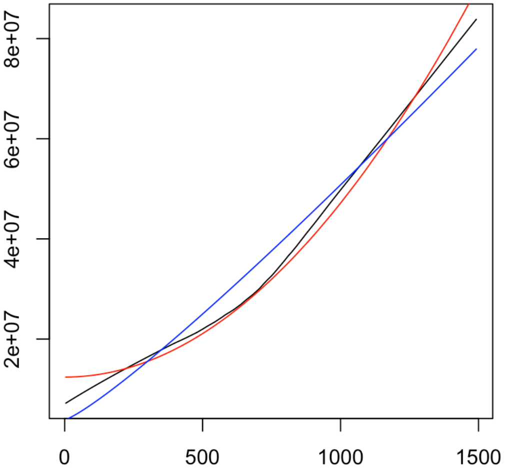
Higher genus

General algorithm given by [2] in $O(g^7 n \log^2 n \log^2 C)$.
In surfaces with higher genus, the decomposition of dual in L_λ is no longer (just) a tree, but might have additional edges.
Self adjusted trees can be modified to preserve the same time complexity.
The time complexity is still $O(N \log n)$, but N is $O(n^2)$.

Dual n^2 case



o^- ∞ o^+
o^+ ∞ o^-
∞ o ∞
∞ o ∞
p_0 0
0 r_0
30
23
22
15
14
27
26
19
18
11
p_{10}
r_{10}
1
1
3
2
q_1 q_2 q_3 q_4 q_5
q_5 q_4 q_3 q_2 q_1



8e+07
6e+07
4e+07
2e+07
0
500
1000
1500

BORRADAILE, G., AND KLEIN, P.
An o(n log n) algorithm for maximum st-flow in a directed planar graph.
J. ACM 56, 2 (Apr. 2009).

CHAMBERS, E. W., ERICKSON, J., AND NAYYERI, A.
Homology flows, cohomology cuts.
SIAM Journal on Computing 41, 6 (2012), 1605–1634.

ERICKSON, J.
Maximum flows and parametric shortest paths in planar graphs.
In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms* (USA, 2010), SODA '10, Society for Industrial and Applied Mathematics, p. 794–804.

FORD, L. R., AND FULKERSON, D. R.
Maximal flow through a network.
Canadian Journal of Mathematics 8 (1956), 399–404.

JOHNSON, D. B., AND VENKATESAN, S. M.
Partition of planar flow networks.
In *24th Annual Symposium on Foundations of Computer Science (sfcs 1983)* (1983), pp. 259–264.